Improving Chrome's Security Architecture

Charlie Reis
Web: Safe to visit any site!
Despite...

- Running untrustworthy code
- Compiled to native code
- Complex formats to parse
- Built in unsafe C++
- With frequently added APIs
There will be bugs

- Finding and fixing bugs is important
  - Fuzzing, VRPs, analysis, etc
  - Automated triage, extensive testing, auto-updates

- Limiting the damage is equally important
System Architecture Matters
Sandboxes reduce bug severity
Sandboxes reduce bug severity
Sandboxes reduce bug severity
Still not a match for web's security model
Site Isolation: Multi-principal architecture
Research → Production
Out-of-process iframes

- Challenging to support web platform
  - Secure compositing
  - Frame proxies
  - State replication
<table>
<thead>
<tr>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accessibility</td>
</tr>
<tr>
<td>Developer tools</td>
</tr>
<tr>
<td>Drag and drop</td>
</tr>
<tr>
<td>Extensions</td>
</tr>
<tr>
<td>Find-in-page</td>
</tr>
<tr>
<td>Focus</td>
</tr>
<tr>
<td>Form autofill</td>
</tr>
<tr>
<td>Fullscreen</td>
</tr>
<tr>
<td>IME</td>
</tr>
<tr>
<td>Input gestures</td>
</tr>
<tr>
<td>JavaScript dialogs</td>
</tr>
<tr>
<td>Mixed content handling</td>
</tr>
<tr>
<td>Multiple monitor and device scale factor</td>
</tr>
<tr>
<td>Password manager</td>
</tr>
<tr>
<td>Pointer Lock API</td>
</tr>
<tr>
<td>Printing</td>
</tr>
<tr>
<td>Task manager</td>
</tr>
<tr>
<td>Resource optimizations</td>
</tr>
<tr>
<td>Malware and phishing detection</td>
</tr>
<tr>
<td>Save page to disk</td>
</tr>
<tr>
<td>Screen Orientation API</td>
</tr>
<tr>
<td>Scroll bubbling</td>
</tr>
<tr>
<td>Session restore</td>
</tr>
<tr>
<td>Spellcheck</td>
</tr>
<tr>
<td>Tooltips</td>
</tr>
<tr>
<td>Unresponsive renderer detector and dialog</td>
</tr>
<tr>
<td>User gesture tracking</td>
</tr>
<tr>
<td>View source</td>
</tr>
<tr>
<td>Visibility APIs</td>
</tr>
<tr>
<td>Webdriver automation</td>
</tr>
<tr>
<td>Zoom</td>
</tr>
</tbody>
</table>
Dedicated renderer processes

Browser Process

Renderer Process: evil.com

Renderer Process: youtube.com
Cross-Origin Read Blocking

- Must allow subresources
- Want to protect sensitive data (HTML, XML, JSON)
- Mislabeled Content-Types
  - Custom sniffing
  - Must allow responses like:

```
Content-Type: text/html

<!-- This is JS. -->
function a() {...}
```

foo.com

Cross-site images, scripts

Cross-site data

```
<img src="bar.com/image.jpg">
<img src="bar.com/secret.html">
```
Site Isolation: Most renderer bugs less harmful

- Shipped on desktop for all sites (2018)
- Shipped on Android for some sites (2019)
  - More memory constraints on mobile

- Compromised renderers can't access most cross-site valuable data!

- Still some tradeoffs and gaps (e.g., Android WebView)
  - Not ready to lower actual severity of renderer compromise bugs yet
Align with OS
Spectre upends assumptions

- CPU's predictive behavior leaks secrets via cache
- Breaks rules of safe languages
  - Can access any address
- No shortage of transient execution attack types
- Works from JavaScript
Difficult to mitigate Spectre in browser

1. **Remove precise timers?** (e.g., SharedArrayBuffers)
   - Not effective: Coarse timers can be amplified
   - Harmful to Web Platform

2. **Compiler/Runtime mitigations?**
   - Not effective: Can't handle all variants
Have to assume access to full address space

- **Site Isolation**
  - Put data worth stealing out of reach
  - Effective for *same-process* variants

- **Align security model with OS/HW enforcements**
  - Hard to trust software boundaries without OS support
  - Reliant on OS/HW mitigations for cross-process variants
Evolve Platform APIs
Push platform towards better security

- HTTPS
  - Encourage adoption
  - Required for powerful features
- Flash deprecation
- Better security APIs
Site Isolation: Constrained by Compatibility

● Site vs Origin
  ○ `https://google.com` vs `https://mail.google.com:443`
  ○ `document.domain` isn't quite gone yet

● Protecting Cross-Site Data
  ○ Have to allow through ambiguous resources, for compatibility
  ○ Not easy to confirm something is JavaScript
Headers, eventually safer defaults

- **Cross-Origin-Opener-Policy**
  - No cross-window scripting. Easier process isolation.

- **Cross-Origin-Resource-Policy**
  - Better hints about what data is accessible cross-origin.

- **Cross-Origin-Embedder-Policy**
  - Enable powerful features (Shared Array Buffers).
  - Don't allow any cross-origin data without opt-in.
Conclusion

- Site Isolation: research to users
  - Compromises needed, but offers best path to protection
- Align security model with OS/HW
- Must push platform forward

Calls to action:
  - Revisit your architectures
  - Help secure the Web